Commonwealth of Massachusetts
Executive Office of Health and Human Services

EOHHS CTO Organization
Information Technology Architecture

Version 2.0

Revision History

Date	Version	Description	Author(s)
Sept 7, 2006	2.0.0.1	Major revision	Robert Skoczylas
Nov. 10, 2006	2.0.0.2	Review and edits	Raoul Sevier
Jan. 16, 2007	2.0.0.3	Review and edits	Raoul Sevier
Apr. 26, 2007	2.0.0.4	Review and edits	Raoul Sevier
Apr. 30, 2007	2.0.0.5	Review and edits	Martin Garden
May 4, 2007	2.0.1	Review and approval	Jason Snyder
May 4, 2007	2.0.2	Additional edits	Jason Snyder

Table of Contents

1. Introduction

Executive Office of Health and Human Services (EOHHS) is transforming its existing Information Systems (IS) architecture and infrastructure from mainframe and stove pipe-based models to that of an enterprise-wide, Web Service driven, Service Oriented Architecture (SOA) model. EOHHS has chosen this direction to capitalize on the advantages afforded by an open, flexible, scalable, service-driven information technology architecture.

The use of systems architecture is recognized as best practice in addressing many types of information systems problems and is, consequently, a necessary precondition for a successful enterprise approach. Additionally, a number of other elements will support the final success. Most notably, the level of understanding of the real business processes in the organization and ability to specify this understanding. Also, organizational support for architecture-driven systems analysis, design and implementation is required for the maturation of a SOA approach.

1.1 Purpose

Information technology organizations are turning to an enterprise technology model to re-align technology with business. By providing an enterprise level architecture for the acquisition and deployment of various types of technology, it becomes possible to achieve the different objectives and fulfill the drivers listed in sections below. In order to achieve the drivers, this document lists the different choices a project development team can make when developing an application and still stay within the boundaries of the architecture.

1.2 Scope

This document is an evolving artifact that will continuously be updated throughout the lifecycle of the SOA program. The conceptualization of the Information Technology Architecture (ITA) is captured in chapters two and three. The chapter four provides architecture domain descriptions that will serve as architectural models for the SOA vision.

In addition to the models this document describes different architectural views for describing the concrete architecture of a service. The views are described in the system architecture document (SAD), which is the document that describes the concrete architecture of the developed service.

1.3 Business Drivers

The IS organization must provide non-IS management with the ability to:

- Access business information faster, cheaper and easier.
- React to changing business and legislative environment
- Accommodate organizational restructuring
- Accommodate business growth and expansion
- Rapidly expose business services to interested parties
- Control the cost of business processes and IS development
- Eliminate reliance on inflexible and unstable existing systems and processes.

- Accommodate existing information technology assets (legacy systems)
- Allow for cost-effective retirement of outdated systems

1.4 Information Technology Drivers

In addition to always keeping the business drivers in focus, the following information technology drivers need to be addressed:

- Consolidate development tools and technologies
- Improve reliability, modularity, extensibility, scalability and stability of existing and new systems
- Establish technical guidance by promoting best practices and standards.
- Exploit new network computing architectures, particularly those based on Web Services standards so that future evolution of the architecture is supported.
- Use packaged applications (COTS) where applicable.
- Replace obsolete and outdated technologies.
- Provide integration mechanisms for non-conformant legacy systems.
- Provide a repository of reusable software components that can be leveraged in projects within HHS.
- Provide integration mechanism for existing system to promote the concept of service-oriented architecture.
- Provide security recommendations in order to protect the HHS infrastructure from unauthorized access.
- Provide solutions that are based on open-standards and vendor independent implementations of those standards and specifications.
- Use Open-Source Software where applicable.
- Establish a process for defining standards.

1.5 Information Technology Architecture

What is the Information Technology Architecture? As its name implies, it describes a plan for different application groups to navigate the common architectural definition being introduced at HHS. Broadly speaking, it is the "grammar behind the prose" that provides an overall script for the development of new service-oriented initiatives. It provides the requisite guidance, relevant anchors, and common constraints for ensuring that individual projects conform to the overall architectural vision for HHS and is based on a blueprint for smart, standard, open, service-oriented services.

The Information Technology Architecture is intended to be used enterprise-wide within HHS. As new initiatives utilize the guidance it provides, it will be changed to reflect new uses and new technological developments. That is, the Information Technology Architecture is a living document, which will serve as the first artifact used by a development organization to kick-start a project. It will provide all the necessary knowledge and references to successfully execute the project.

A single unified architectural vision and realization is imperative for HHS to transform its computing infrastructure to a highly flexible, service-driven, information utility model and successfully take on a variety of business challenges. This model is evolutionary and will be delivered iteratively.

1.6 Information Technology Architecture Objectives

The Information Technology Architecture is the fulcrum by which new initiatives leverage the new architecture, process, and organization, ensuring that every initiative across the enterprise achieves consistently high quality. The following are the objectives of the HHS ITA:

- Define a conceptual model for the ITA itself and show how it ties process, organization and architecture into a living and continually evolving structure.
- Define HHS Enterprise target architecture with associated architectural views.
- Define HHS Application level architectural/design views that all applications will be expected to provide.
- Describe how the ITA will be maintained.

2. Architectural Principles for ITA

The Information Technology Architecture provides a standard technical architecture for all EOHHS technical business and shared services so that they are able to rapidly realize the business drivers. In this context, it relies on a set of architectural principles that provide a stable foundation upon which important service design and implementation decisions can be made. These principles describe the foundation assumptions on which the services are built and help identify the steps that need to be taken to move from the current state to an ideal future state.

It focuses primarily on principles that govern the implementation of the architecture, establishing the tenets and related guidance for designing and developing information systems.

2.1 Technology Architecture Principles

Think business process/service, not system/application

Orient service development around business processes so that it is possible to execute these processes from the business models directly through a process engine and also measure, monitor and administer processes dynamically

Adopt Model Driven Architecture

Insulate application architecture from the underlying technology platform by moving in the direction of a model driven architecture.

Define Domain Frameworks

Adopt domain frameworks so that new features can be added by using a few new components with no changes to existing code

Use hybrid approach for legacy management

Adopt a hybrid approach for legacy management involving both componentization and encapsulation of existing functionality, and integration of conformant systems and services.

Design for Service Quality

Use workload models based on various factors to scale systems in order to meet present and anticipated Quality of Service (or Service Level) requirements.

Build for collaboration

Adopt technologies to facilitate inter-enterprise collaboration and process automation.

Adopt open and standards-based design

Adopt process and service standards to harmonize processes of the business domain and streamline interactions

Treat data as a shared asset

Adopt technologies that allow data to be shared consistently across the enterprise, with relevant users having access to the data necessary to perform their duties

Secure Information

Implement security measures to protect information at the network, container, and service (code) levels in order to provide for the Confidentiality, Integrity, and Availability of the data.

2.2 Quality of Service

The above principles guide the technical architecture of information systems and applications that ultimately help realize the business drivers. Hence the architectural principle should satisfy one or more Quality of Service parameters (QoS) that help realize the strategic priorities and business drivers. The following QoS factors have been identified based on industry research and the architectural principles covered in the previous section:

- Performance – ability to deliver results (throughput or bandwidth) within the least response time (latency)
- Scalability – ability to cater to greater demands imposed upon the system (e.g.: support increased number of users, products) without affecting any of the other QoS parameters
- Reliability – ability to function with the least occurrence of failure
- Availability – ability to maximize the time when the system is available for use
- Securability – ability to authenticate and authorize users to provide secure access to the system in a traceable (auditable) manner
- Manageability – ability to monitor and configure systems easily and detect operational characteristics related to performance and failures (remotely).
- Maintainability – ability to modify the system easily, with the minimum amount of work or rework over the life cycle of the application
- Extensibility – ability to make significant enhancements or changes easily.
- Usability – ability to allow users to use and navigate the system easily.
- Serviceability – ability to be repaired or updated easily and rapidly without affecting reliability or availability of the system.
- Reusability – ability to use individual components or services in the building of unrelated modules or services
- Interoperability - ability of components to work with each other regardless of their underlying platform.

2.2 Strategic Priorities

The selection of an architectural style was based on the capability to meet the business drivers as well as to fulfill the QoS requirements. In addition, the conceptual architecture must be aligned with the prevalent industry standards to ensure vendor neutrality, and protect the investment from untimely obsolescence. Component based Service Oriented Architecture best meets all of these criteria. It also allows the plug and play of functionality from the best of breed applications, and leverages the investments made in legacy applications.

The alternatives to Service Oriented Architecture are many and diverse, but the one trait shared by all of these is that they are all proprietary and package driven, and often a monolithic architecture, lacking the componentization and the flexibility offered by Service Oriented Architectures. Further, these alternatives will be based largely on the platform of a single vendor whose direction may diverge from the goals of the organization in the future.

The following chapter describes the characteristics of a component based Service Oriented Architecture.

3. Service Oriented Architecture

Enterprise IT infrastructure today consists of what are generally referred to as "silos", or "stovepipes". Over the years, an organization evolves a myriad of systems, each typically chartered and funded by a single line of business with a narrowly focused automation goal. Each of the resulting systems, developed by different groups within the organization, often using differing technologies and platforms, are typically integrated, if at all, in a piecemeal, point-to-point fashion. This is usually facilitated via the exchange of data files rather than real-time transactions. These silos are sometimes combined or "integrated" into "silos of silos" that are even more difficult to manage. Systems that are largely fractured and isolated must somehow be brought together into a cohesive complex providing seamless access to all of the state's processing capabilities.

An effective approach to achieving integration and enabling the rapid development of new business offerings is to interpose a "services tier" between client devices and back-end resources. Through this approach, back-end resource functionality is made available via well-defined, network-callable components. These components can make back-end functionality readily available to clients, but unlike the silo approach, provide building blocks for combining simple services into fully featured, value-added services.

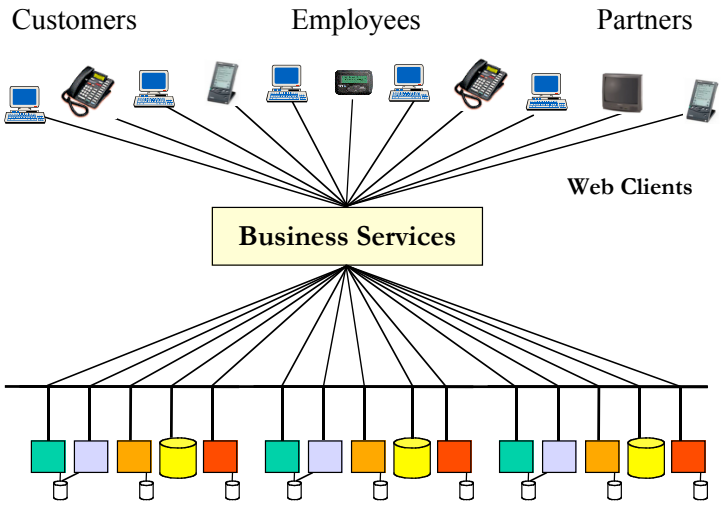
The service-oriented architecture (SOA) establishes a standard middle-tier platform across the enterprise that allows for easy discovery and integration of information resources. Services-oriented architecture involves standardizing middle-tier interface technology such that once implemented, a middle-tier component is available for use by other middle-tier components without complex message reformatting and protocol conversions. Such standardization transforms the middle-tier from being a collection of silos and elevates it instead to being an enterprise-wide services tier.

The rationale for choosing the SOA architectural style can be summarized as follows:

- It embodies best practices for the technology to be used at HHS
- It satisfies short- and long-term architectural needs of HHS
- It reduces risks associated with ambiguous specifications and rediscovery of standard patterns
- It makes divide-and-conquer approach in the evolution of existing systems practical
- It provides the foundation for integration between various enterprise services

The important consequences of adoption of that architectural style are the following:

- It is the interaction of services and assembly of components that creates "applications" and "systems" as opposed to a singular monolithic application
- Distribution and scalability is built-in and not an afterthought.



3.1 Attributes of Service Oriented Architecture

A service-oriented architecture is an architectural style that emphasizes functionality built using standard component-container technologies. Conceptually, the service-oriented architecture is unified yet it is not monolithic, that is, it is a collection of coarse grained, loosely coupled business services and has the following important characteristics:

Service-based, Not Just Code-Based*:* The unit of re-use should emphasize reusable generic service functionality, be readily available, deployed as components across the enterprise and capable of being centrally managed and controlled to achieve the desired quality-of-service (QOS) levels.

Assembled, Not Built: Using services as building blocks, systems should be formed, to the extent that it is possible, by assembling existing services, not by starting from scratch. Some of the parts may even be provided by resources from outside the corporate boundaries. Assembly should be rapid and efficient, not a costly process of custom integration. This however does not preclude from building the service blocks from beginning if it makes business sense to do so.

Physically Distributed: It should be possible to distribute processing across multiple physical network devices. This allows spikes in demand to be met using horizontal scaling techniques.

Quality of Service (QoS) Driven: The services framework must be implemented in a manner which takes into account management, security and other systemic qualities.

Implemented in Layers: Layering maximizes the independence of processing components from their underlying platform implementations.

Layers provide a way for *logical* organization of components in the system from the point of view of their technical – rather than business-related – functions. In a layered organization of a system, the emphasis is placed on the following:

- Reduction of interactions between layers - the typical rule is that a given layer interacts only with its immediate neighbors
- Reduction of dependencies between layers – a given layer is ideally directly dependent upon a single layer that provides services to it.

In addition to providing a clean logical decomposition of the system (in addition to components), layering reduces impact of changes or replacement of frameworks used to construct a given layer.

Functionally isolated: Functionality should be partitioned according to logical function, enabling services to evolve independently of one another in response to ever-changing requirements without impacting other services.

3.2 Components and System Decomposition

The main building block of the SOA is a software component or service. Software components are units of composition with contractually specified interfaces and explicit context dependencies. A component has the following attributes:

- It provides services using well-defined interfaces, ports and protocols
- It encapsulates state and behavior
- It depends only on a component framework or operating system to be started up and to communicate with other components
- It can be constructed, tested, and deployed independently of other components

In general, there are two types of components:

1. Functional components – they realize designated business functionality

2. Infrastructure components – they do not introduce any business-specific functions but are required for functioning of the system (and are still independently deployable components) or support of enterprise concerns such as auditing or monitoring.

The approach for determining the scope of top-level components is based on the following:

- High cohesion of functions within a given component – as e.g. in case of a security component or a client management component

- External usability of a service – the determination of functional components is to include consideration of systems external to a given application that may use facilities provided by these components in order to execute specific parts of the overall business process.

- Reduction of the overall number of components through abstraction and refactoring – For example, components in an application should consider abstracting small differences between related functions in a number of areas to provide a common customizable service or its reusable implementation. This effort, however, will be limited in practice by delivery timeframes and availability of business specifications from potential clients outside the application project. Refactoring existing code may be separate from service delivery.

Given that the architecture uses J2EE technology, software components are implemented as Enterprise Java Beans (EJBs) or Servlets. This choice leads to the following view of the overall system:

- The actual implementation and functionality of an EJB or Servlet component is obtained from interaction of a number of instances of Java classes.
- A business service is provided either by a single physical component or a group of interacting components.

The main elements of system decomposition are classified in the following:

- Functional components
- Infrastructure and ancillary components

The system will also utilize a number of frameworks (persistence, presentation, workflow, etc…) and even though those frameworks will be important *parts* of the system they are not *components* in the sense discussed above.

The rationale for the above form of decomposition is the following:

- Focusing on functional services creates and architecture that can be understood by non-technical people
- It facilitates mapping between business processes and technological infrastructure that supports those processes
- It provides a natural unit of encapsulation that reduces dependencies and unnecessary interactions with other services or systems: service is the main unit of public reuse (follows functional decoupling)

3.3 Definition of a Service

While the functionality of a Service varies considerably depending on the category of the Service or what layer the Service resides in, services in general share a common set of characteristics:

Well-defined Public Interfaces: Services expose a well-known, network-callable remote interface to its potential users. This allows them to be used by other services and to deliver their functionality across different networks and access devices. The service is described using a standard description language and uses a service registry for storage of service description. An example of such implementation would include Web Service Description Language (WSDL) for

describing the service and a Universal Description and Discovery Interface (UDDI) directory as the broker where service descriptions are stored.

Shared and Reusable: Services are not tied solely to one application or even a module. Rather a service in itself is a complete and standalone entity that can be assembled with other services or loosely integrated into a distributed application.

Location and System-Independent: Services are not tied to a specific platform or a deployment location. This allows for different deployment topologies that further address changing business environments.

Predictable Quality-of-Service: Services possess performance, scalability and other systemic qualities that are well known. This allows, for example, planning runtime environments based on known performance characteristics.

Aggregation: Services can be decomposed and recombined to form new services as required to offer new functionality to meet new business requirements.

3.4 Key Benefits to HHS

While the total benefits of service-oriented architecture are substantial to any organization whose enterprise is built along these concepts, some of its attributes are more significant and compelling than others. Based on the issues that have faced HHS in the past and present and the challenges that it needs to overcome in the future, certain key benefits of service- oriented architecture have emerged.

Consistency in the way the system as a whole responds to meet the quality of service requirements will be a key benefit along with its ability to seamlessly integrate the multitude of legacy applications in HHS. Because of the nature of the business HHS is involved in, as well as the organizationally and functionally distributed business environment, the location independent and functionally decoupled attributes of a service-oriented architecture will allow flexibility in how systems are deployed.

One must understand that having an architecture is not enough. An organization has to invest in the whole process in order to have a successful delivery of the architecture. It is the responsibility of EOHHS to maintain the latest architecture documentation, inventory of tools, standards and catalog of reusable components and services. It is critical to develop widely adoptable services to solve common business patterns within HHS and allow new service development efforts to focus on the more unique business aspects.

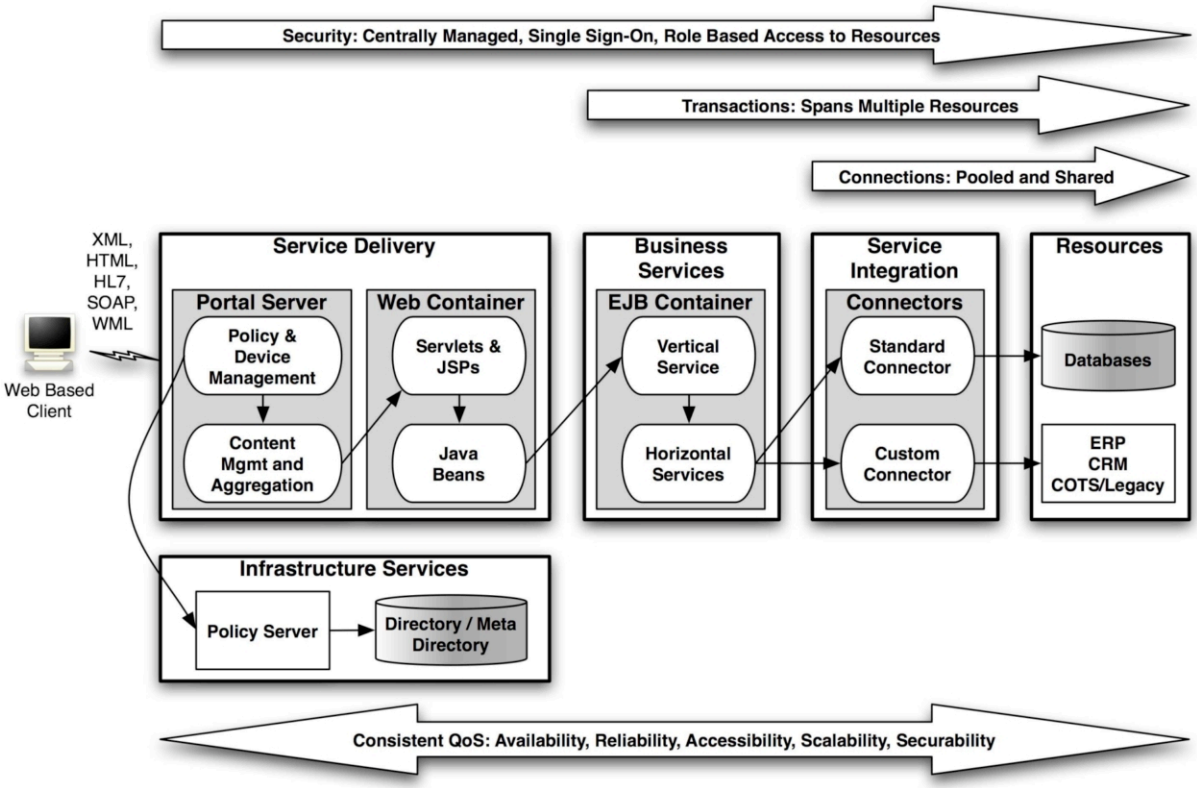
The ITA should drive decision making from the technology perspective and the Unified Process Methodology from a software development life cycle perspective. The HHS UP Methodology User guide provides more details.

3.4.1 Consistent Quality of Service (QoS)

The mechanisms that realize the quality of service are incorporated into the service layer that sits between the users and the enterprise resources. The non-uniformity in the quality of service across the enterprise

resources is shielded by the middle service tier, which applies mechanisms to ensure that any inconsistencies are compensated.

The figure below illustrates some of the key mechanisms that are used to ensure that each user experiences the same level of service regardless of which service the person accesses. Specifically, a user has a single point of access to interact with the services, which is the logical Service Delivery tier. The services in this group checks the credentials of the users and assigns privileges or entitlements that are used throughout the system to permit access to those services that the user is authorized to invoke. Reliable transactions are also ensured as the service tier controls the transactions across disparate enterprise resources. Availability and scalability are also enhanced as the service tier utilizes such mechanisms as resource and connection pooling to ensure that precious resources are shared instead of created for each request.



3.4.2 Integration of COTS Legacy Applications

The capability to seamlessly integrate COTS and legacy application is another key benefit of the service-oriented architecture for HHS. Within the Service Integration layer, standard and custom connectors allow these applications to be treated as just another enterprise resource. With respect to COTS applications, JCA compliant connectors supplied by the product vendors themselves or third party connection adapter vendors can be used by the business services to access these resources while preserving the transaction and security context. For legacy applications, custom connectors can be

developed that allow interaction between the business services and the legacy application. If these custom connectors are developed in accordance with the JCA specification, the same preservation of transactional and security services can be achieved. The leading J2EE application server vendors currently offer development tools to build such JCA compliant connectors.

With the availability of custom and JCA connectors, business services can retrieve capabilities offered by the COTS and legacy applications and merge, translate or enhance the functionality. Further, applications can be included in the new architecture as part of a phased deployment strategy.

4. Information Technology Architecture

The Information Technology Architecture (ITA) is described through a set of views, each from the perspective of a different stakeholder. An individual view captures items meaningful to the stakeholders as elements and their interrelationships expressed in a standard form, the structure of the view, and the view's correlation with other views

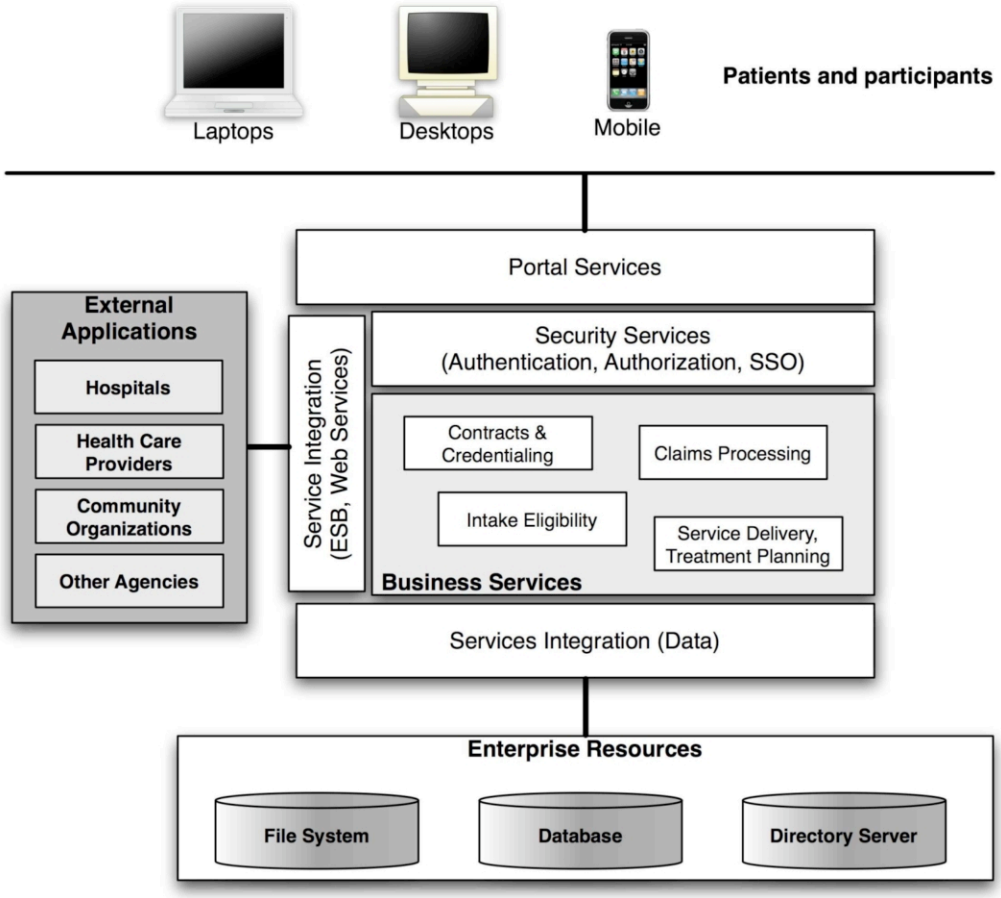
4.1 Conceptual View

The Conceptual View serves to highlight the distinguishing concepts of the architecture. As illustrated in the Conceptual View figure, the ITA is composed of a collection of functionally decoupled business services residing within a unified architecture. At the core of the model reside the business services instead of a collection of applications. They encapsulate disparate business functionalities yet share a common substrate and semantics. Since they are decoupled from one another, services can be deployed in whatever configurations necessary to meet business needs. Further, as business requirements evolve, so can the business services, as new services can be rapidly *assembled* instead of being developed or purchased.

Typically, groups within the organization possess a mixture of Commercial Off-The-Shelf (COTS), frameworks and custom-developed legacy applications as well as data stored in databases and directories. In such an environment, functionality and data embedded in one application cannot be easily merged with functionality and data provided by another. Such an endeavor usually requires the creation of a yet another separate application. Further, each of these applications relies on a proprietary client for user interaction.

The business services may retrieve and integrate information residing in backend enterprise resources such as databases and legacy applications through a common Service Integration layer. The Service Integration layer standardizes the method and the semantics of accessing the enterprise resources. Once the business service has retrieved the information from the enterprise resources, they can aggregate this information and expose the results as a network accessible service to the users. These services are then delivered to the users through a common delivery channel regardless of the type of client device. This common Service Delivery channel provides a single point of access to the business services and ensures that centralized security and context policies are applied to each user.

A unified, service oriented architecture with a centralized delivery channel and a common integration layer enables the realization of consistent quality of service across the enterprise. Aspects such as scalability, security, availability, reliability and maintainability can be readily predicted, easily controlled and managed.



Conceptual View of the architecture

4.2 Logical View

In the Logical View, the architecture is decomposed into parts that address the different functionality areas. It also describes the relationship among these parts and how they interact with one another.

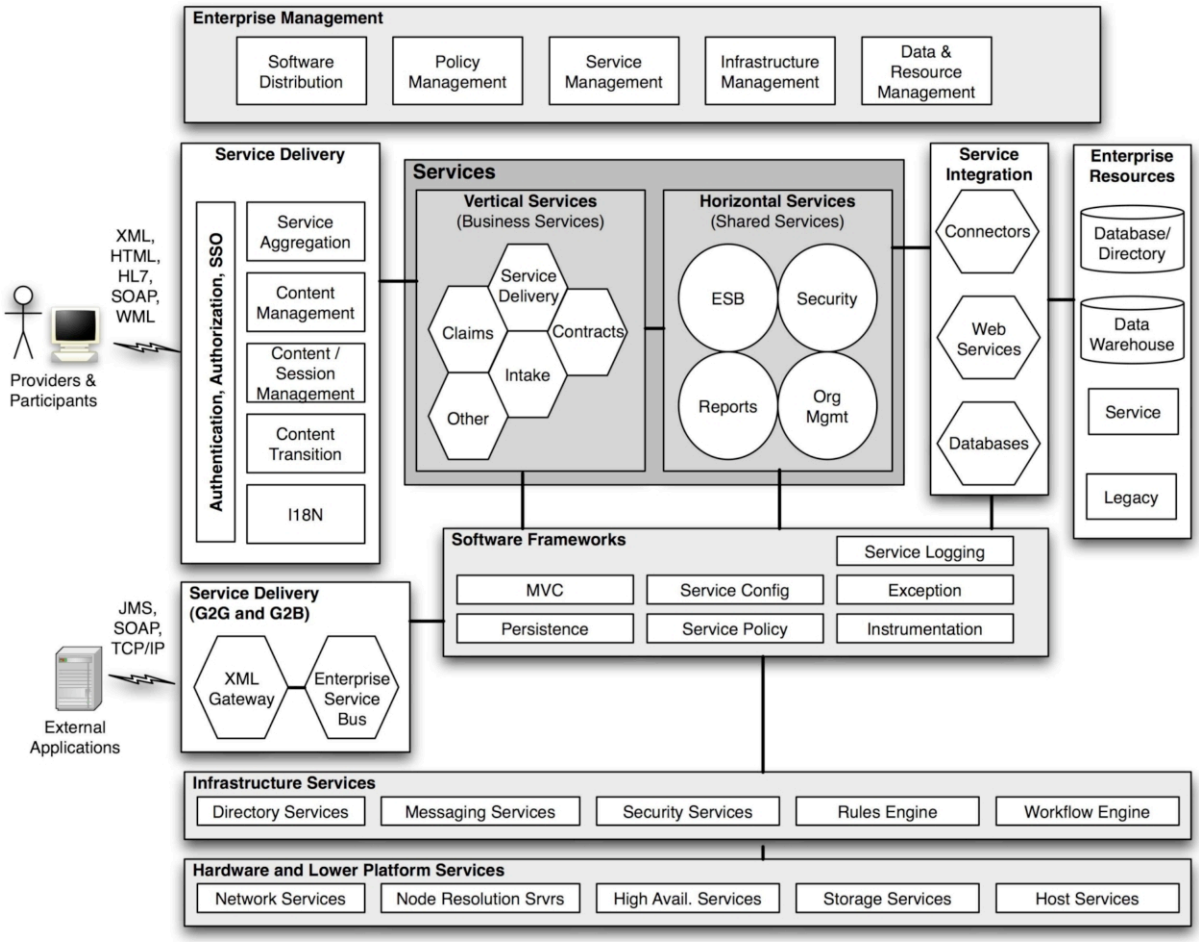
4.2.1 Overview - Taxonomy of IT Services

The organization and categorization of a set of unrelated objects is known as taxonomy. It is convenient to group services by their functional area, and at the highest level the initial set of groups (services taxonomy) is as follows:

- Frameworks
- Service Delivery
- Business Services
- Infrastructure Services
- Service Integration

- Hardware and Lower Platform
- Enterprise Management

The figure below shows how these service groups are related and highlights the significant or representative services within each group.



4.2.2 Frameworks

Frameworks are generic, typically business-agnostic reusable components and libraries. The core function of the business services is to deliver some business functionality. However, these business services must also perform some common, non-business-specific tasks, for example to provide case management functions, manage the Software Development Life Cycle, initialize a database connection or to perform monitoring and management functions. The extraction of these common functionalities from the services and utilizing them as a shared framework across the services increases architect, designer, and programmer efficiency. A shared framework also offers consistent and predictable quality of service throughout the system. Widespread use of common software frameworks imposes a common programming style across applications, enabling greater programmer mobility within the enterprise.

Frameworks can also improve portability and protect the application components from instability in the upper platform layer (i.e., vendor specific Web servers and application servers). When the frameworks are developed based on the capabilities of standards-compliant virtual platforms (e.g. Servlets, JSP, EJB), changing from one vendor to another will not have dramatic impacts, particularly if the frameworks are designed to shield vendor specific implementation details.

This set of frameworks address basic functionality that will be needed by all initiatives and are discussed briefly in subsections below. The frameworks are categorized into Application and Infrastructure frameworks.

Application Frameworks

Application frameworks are independent of the infrastructure platforms while providing non-business specific functionality.

Service Logging

Achieving manageability goals requires producing an accurate trace of application activity in the form of a streaming log. These logs are used for operations management, auditing, and troubleshooting. They typically capture information such as security failures, configuration errors, performance information, and bugs encountered in the application or platform. Well-designed and consistent logging facilitates software servicing and maintenance at customer sites by producing log reports suitable for analysis by end users, system administrators, field service engineers, and software development teams.

Service Configuration

Every application requires customization at some level to indicate desired settings for variant facets. Examples include directory locations, switch settings, level selections, and component lists. A common declarative approach to storing and accessing these variant facets based on XML configuration files will ensure a level of consistency for application configuration and deployment. Further, the use of XML enables the use of industry-standard tools for transformation and manipulation of application configuration files.

Service Exceptions

One important issue that must be addressed by all Java programmers is exception handling. The ITA must establish an enterprise-wide approach to exception representation and processing to avoid duplication of effort between application groups, and also to ensure a consistent level of response to operational problems. That is, exception handling is typically given short shrift by designers and programmers because it is time consuming and difficult to handle well. Providing exception handling in the form of an enterprise-wide framework will mitigate the risk that application groups will leave it to the end of the design and development cycles, avoiding the concomitant danger that exceptions are not handled at all. Furthermore, exception handling will be integrated with the logging and event frameworks for recording and notification.

Events and Messaging

A common requirement for applications is enterprise-level messaging. When something happens in one area, other areas may need to be informed of it, sometimes across application boundaries. The notification can be propagated synchronously or asynchronously depending on the circumstances. Further, the originator of the event may or may not need to know the receiving parties.

Service Registration and Discovery

Services must expose themselves to the world in some way. The standard way of doing it is to publish their availability in a directory server, analogous to yellow pages. Obviously, for ease of management and development, the fewer directory servers there are and the more transparent they are, the easier it is for system administrators and developers. Service registration can be accomplished through established standards such as Lightweight Directory Access Protocol (LDAP) via JNDI. In the future this may evolve to include emerging standards such as Universal Description, Discovery, and Integration (UDDI). In both cases, Java APIs exist for accessing the underlying mechanisms (JNDI for LDAP and JAXR for UDDI).

Service Instrumentation

A number of solutions for systems management is available, based on standards such as Simple Network Management Protocol (SNMP), Desktop Management Interface (DMI), and Common Management Information Protocol (CMIP), as well as other more proprietary protocols. In 1996, the Internet Distributed Management Task Force initiated the Web-Based Enterprise Management (WBEM) effort, envisioned as a single web-based standard that integrates the other standards. WBEM and closely related Common Information Model (CIM) are now well established as DMTF standards, and most major vendors are adapting their solutions for compatibility.

In the Java community, Java Management Extensions (JMX) is being developed as a common API for accessing management information spanning from the lowest level hardware component to the proprietary application components. Defined on top of standards such as WBEM/CIM, JMX provides the tools for building distributed Web-based, modular and dynamic solutions for managing devices, applications and service-driven networks.

Model-View-Controller

The idea behind Model-View-Controller has its roots in the Smalltalk language environment. The goal of this framework is to achieve decoupling among the software components that are responsible for encapsulating business functions, rendering the content and controlling the navigation or flow. This fundamental concept can be applied to a collection of Servlets, JSPs, JavaBeans and EJBs to achieve architecturally decoupled J2EE systems. JSPs are used to represents Views, Servlets as Controllers and JavaBeans and EJB's as the Models.

Infrastructure Frameworks

System frameworks perform low-level functionality on behalf of the caller and interact with the infrastructure services to realize the request. They serve as bridge between the business service and the infrastructure service.

Service Policy

At least two aspects of security are common across all applications. As users access the services, they must be accurately identified (authentication) and their permitted access levels must be effectively managed (authorization). Using a common framework for these two security activities will reduce the burden on application development teams.

Session Management

Session management manages the state of an application as a server-side object. Session management deals with clients and the conversational state that is built up in the course of a conversation. It must be kept in an appropriate location, based on the requirements of the application in terms of whether a user can transparently re-establish their context with the system. Thus, there are a variety of options. A framework should handle each option that needs to be supported, so that it is done once and only once. Persistence deals with storing state with guarantees of integrity, consistency, and durability.

Transaction Management

A service driven architecture must address the familiar information technology problem of transaction management to ensure the atomicity, consistency, isolation, and durability (ACID) requirements of applications. The Java Transaction API (JTA) of J2EE provides a solid basis for a common transaction management framework. J2EE also includes the feature to delegate the responsibility of transaction management to the underlying J2EE platform, which allows only the business logic to be include in the application components.

Persistence

Business objects manipulated by services must be procured and stored in permanent data storage, usually realized as relational databases. A common approach to object persistence using either EJB Container Managed Persistence (CMP) or a third-party persistence mechanism must be selected. If persistence is required in the Presentation Tier for sites that rely only on Servlet and JSP, a common data source location and resource-pooling framework is required.

Directly related to Object Persistence, caching of frequently accessed, read-only persistent objects (e.g. code tables, option descriptions) at the application server level can greatly enhance application performance. A common framework for loading and refreshing these reference caches will ensure that application teams can utilize this performance enhancement

Service Context

Service Context is adequately detecting and managing the client device profile to enable proper presentation of service results. Therefore, whether a consumer is accessing a Web service from a WAP-enabled phone, MIDP hand-held device, Internet browser, or rich client should not affect any service processing other than the presentation layer.

4.2.3 Service Delivery

The information the Business Services provides to it users must be translated, formatted and delivered based on the capabilities of the device the user utilizes to access the system. Service delivery addresses the way in which services are presented to users. Services can be tailored to a specific user, based on a number of factors, such as the device they are using for access, their location, their indicated preferences, or the type of user they are. All of this is done in conformance with the designated policies. Further,

services from the enterprise may need to be aggregated into larger units, presenting the user with a diversity of services on demand.

4.2.4 Business Services

Services encapsulate the high-level business functionality and offer these capabilities to the users and other services. In the HHS architecture, services are categorized as Vertical (or Business) Services and Horizontal (or Shared) Services. The distinction is made depending on the scope of the service. Horizontal service is a micro-service that performs a specific functionality related to infrastructure. Vertical service is a business component that represents a "macro-service" such that it provides a complete set of functionality related to business processing (E.g. Intake Eligibility and enrollment). Vertical service could be a coarse-grained component, which talks to other business components to form a functional public service.

Horizontal Business Services

Horizontal Business Services do not directly offer business functionality to end-users. Rather, they are meant to encapsulate common business functionality that may be present across different vertical business services. Hence, horizontal business service are not "owned" by a specific business unit but instead are shared by many organizations.

Vertical Business Services

Vertical business services encapsulate core, high-level business functionality such as Contracts and Credentialing, Claims Processing, Eligibility, Enrolment and Service Delivery that are offered by the various organizational units within HHS.

4.2.5 Infrastructure Service

Infrastructure services provide system level support services for the business services. They are the foundational mechanisms of technology that allow other services to be built. Typically, they address areas such as directory interaction, security, messaging and state management.

Directory/Meta Directory Service

The Directory/Meta directory service stores enterprise data in a hierarchical relationship, typically to support read only operations such as for authenticating and/or authorizing against stored user credentials.

Messaging Service

The Messaging Service provides the core queue infrastructure and programming extensions (JMS API support) to be able to provide asynchronous capabilities in the application architecture.

Security Service

The Security service uses the directory/meta directory and policy services to authenticate and/or authorize users. It also collects the entitlements based on the credentials stored in the policy server to provide fine-grained security features in the application architecture.

Email Service

The Email service provides a centralized mechanism for exchange of information through the sending and receiving of mail messages. Some of the collaboration services like scheduling meeting, appointments and calendar services can also be accomplished using email service.

Transaction Management Service

The Transaction management service handles transaction life cycles to a particular resource like database or an ERP system and across multiple resources, which might involve complex scenarios like two phase commits. The J2EE application server provides this as an out of the box functionality.

Resource Management

Resource management service includes pooling a set of costly resources like Database connections, instances of business service component, threads and allocating them based on request/demand. It is a very critical part of any application architecture as the management of these resources has a very large impact on the Quality of Service (QoS) goals of the system. J2EE compliant application servers are mandated to provide this functionality out of the box.

Application Clustering Service

Application clustering service involves distributing the application instances on multiple hardware platforms having the same infrastructure set up. The Clustering service aims at balancing the load on the system by distributing it horizontally across instances that are tied to together as a cluster so that they share their state and session information. J2EE application servers support and allow clustered mode of deployment and there are standard best practices that have been developed to be followed for components that are deployed in clustered mode.

Persistence Service

Persistence service provides the access layer over the data resources like the database products for the business service components that are deployed in the J2EE application server. Persistence service can be custom developed or realized using COTS products.

Rules Engine

Rules engine extracts the business specific information and stores it in a configuration that is referenced by the business services. They move the logic driven rules to configuration from code so that when business drivers change it must be a matter of altering the rules through the rules engine and not the code.

Workflow Engine

Workflow Engine is the Process manager component that handles the decision flow and process flow mechanism between services. Most common application server vendors provide a process management module that lets you design the decision logic that flows through the business service components deployed in the application server.

4.2.6 Service Integration

Business services access enterprise data, interact with COTS packages and invoke legacy applications in order to fulfill the user's request. Service integration provides the business services with the capability to

interface with these Enterprise Resources using mechanisms such as a message bus, custom and standard compliant connectors.

Custom Connectors

Custom connectors are components or a collection of components that provides a set of proprietary or custom developed APIs for accessing Enterprise Resources such as ERP, CRM or legacy systems.

Messaging

Messaging in the context of service integration refers to a collection of components that provide a set of APIs for the synchronous and asynchronous transfer of data to external business partners or to Enterprise Resource such as legacy systems.

Standard Connectors

Standard connectors refer to components or a collection of components that provide a set of APIs to communicate with resources that expose standards compliant interfaces such as Java Database Connectivity (JDBC) and J2EE Connector Architecture (JCA).

Hardware and Lower Platform

Underlying all of the other service taxonomies, Hardware and Lower Platform provides the processing, storage, and network hardware, as well as the operating system and networking protocols.

Networking Service

Network services provide the capability to connect, transfer, transform, and route hosts or clients on LANs/WANs in order to support information exchange. Transfer data types can include Wireless, Voice, and Video.

Name Resolution Service

Name resolution services provide host and data name resolution for network resources. They include services such as DNS, WINS, and TNS Names Server.

High Availability/Performance Service

This category of services provides high availability and performance improvements by distributing access and load of hosts and/or clients. Includes services such as content switches, content engines, hardware based SSL accelerators, and network load balancers.

Storage Service

Storage services provide random and/or sequential data access. They include services such as SAN, NAS, disk subsystems, tape subsystems, optical disk subsystems, and flash memory devices.

Host Services

Host services include the hardware platforms on which all of the IT services and Enterprise Resources reside.

4.2.7 Enterprise Management

All aspects of an enterprise must be managed to ensure its smooth operation. Enterprise management includes all of the activities related to the monitoring and management of enterprise assets including the automated, network enabled distribution of applications and components. Policies for access, authentication, logging and other areas must also be managed. Business Services and infrastructure mechanisms must also be monitored for availability and performance. The same is also true for Enterprise Resources.

Software Distribution

Software distribution service provides the capability to manage the dissemination of executable code.

Security Policy Management

Security policy management provides the capability to monitor and enforce the adherence of authentication, authorization, encryption and auditing guidelines and best practices across the enterprise.

Service Management

Service management provides the capability to monitor and administer application level business services.

Infrastructure Management

Infrastructure management provides the capability to monitor and administer hardware platforms as well as web and application servers, messaging servers and other virtual platforms.

Data Management

Data management provides the capability to monitor and administer databases and directory servers.

Network Management

Network management provides the capability to monitor and administer networking hardware such as routers, bridges and switches as well as storage devices.

Backup/Recovery

This category of service provides the capability to perform redundancy operations and restoration of data.

Configuration Management

Configuration management provides the capability to perform version control and the tracking of changes to resources such as source code and project documents.

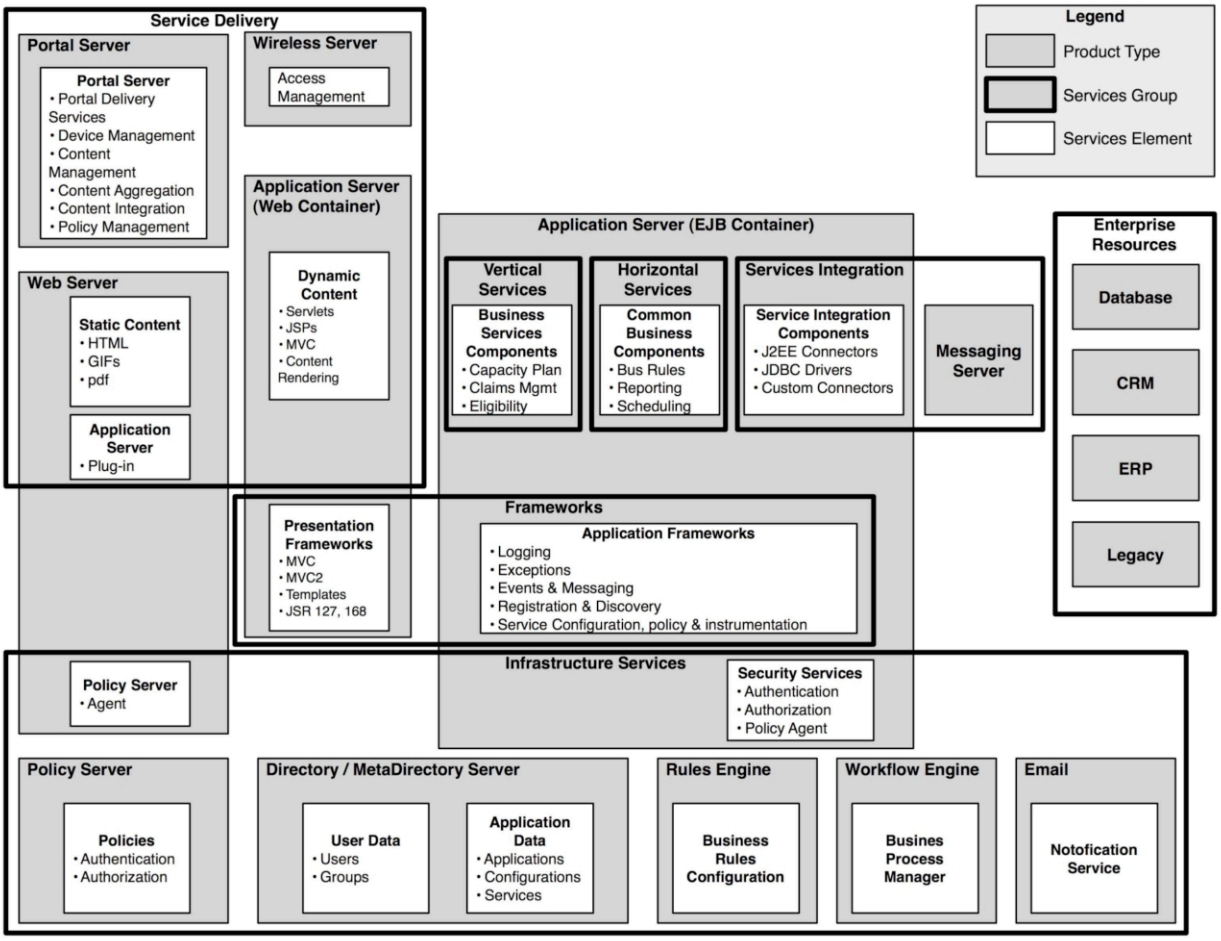
4.3 Process View

This section elaborates on each of the elements of the ITA and maps them to the process, or execution environment, that they will run in. As indicated, each of the Taxonomy groupings consists of several elements that may span multiple processes.

As shown, the main processes will be:

- Web Server
- Application Server
- Directory Server

- Policy Server
- Messaging Server
- Database



4.3.1 Web Server

The Web Server is the infrastructure that handles the following

- Serving of static HTML pages
- Serving image files (JPG/GIF)
- Serving document files (PDF)

The Web Server infrastructure also comes in with a plug-in to invoke the Servlet/JSP container that resides in the infrastructure of the J2EE Application server or the Web server to handle presentation tier functionalities.

4.3.2 J2EE Application Server

The J2EE Application Server is the platform designed for enterprise customers and independent software developers who require high quality of service and performance for mission critical e-commerce applications. It is used to build and deliver application services to a broad range of servers, clients and devices and to integrate with business-to-business applications.

The J2EE Application Server offers a host of value added features that extend the base capabilities and services offered by supporting the standard APIs without compromising the basic standards compliance customers rely on. They provide support for Enterprise Java Beans, Servlets, Java Server Pages, Java Mail, Java Activation Framework, Java Transaction API, Java Transaction Service, Java Messaging Service, Java Naming and Directory Service, JDBC, RMI/IIOP, Corba, Java API for XML Processing, and Web Services. The two major components of a J2EE application server are the EJB container and Web Container.

Web Container

The Web Container provides support for presentation components like Java Servlets and JSP. Servlets act as the Controller that manages the incoming HTTP requests and JSPs are used for the content aggregation and presentation. The Web container also hosts open standards based frameworks like Apache Struts and MVC components.

EJB Container

The EJB Container provides the basic infrastructure and support for hosting the Enterprise Java Beans (EJB) that perform most of the business functionality in J2EE environment. They also provide lifecycle management of these components and manage transactions, security, and persistence. EJB containers also provide the environment for integration with external legacy applications like the ERP, CRM and Mainframes by supporting the J2EE Connector Architecture (JCA).

4.3.3 Policy Server

The Policy server infrastructure provides storage capability for user/roles information and the eligible entitlements for the respective roles. This information will be used to provide "fine-grained" access control to the various business and application functionalities at a component level in the J2EE platform. The role information stored in this infrastructure can be propagated across tiers and hence a consistent implementation of security could be achieved.

4.3.4 Directory/Meta Directory Server

The Directory Server is the infrastructure that provides a central repository for storing and managing identity profiles, access privileges and application and network resource information. Information stored in the Directory Server can be used for the authentication and authorization of users to enable secure access to enterprise and Internet services and applications. It helps improve security and protection of key corporate information assets by ensuring appropriate access control policies are enforced across all communities, applications, and services on a global basis.

4.3.5 Messaging Server

The Messaging Servers provide the core infrastructure for managing messages and the organization of message destinations like Queues/Topics and its associated messages. They also provide access APIs for interacting with core messaging infrastructure. In the J2EE platform, Java Message Service (JMS) is the standard API for the Java components to interact with the messaging server. Most popular messaging servers like SonicMQ, Sun ONE MQ, and IBM MQ Series provide support for JMS to be able to send and receive messages.

4.3.6 Database

Database is the infrastructure that enables storage of application and business information. It resides in the resources tier in the J2EE platform as a repository for storing all business related data. Databases provide drivers which act as the mechanism to access them externally. In the J2EE platform the JDBC API defines the access mechanism for the databases. Most of the commercial vendors such as Oracle, Sybase and IBM DB2 provide JDBC support.

4.3.7 Workflow Engine

Workflow Engine is the infrastructure that provides process automation capabilities in the J2EE platform. The workflow engine helps in designing the process flow between the business components using standard GUI. In the J2EE realm, most of the business components are modeled as services and the workflow engine manages the informational and logical flow between these services to accomplish a complete business process. Some widely used workflow engines include Weblogic process manager and Versata.

4.3.8 Rules Engine

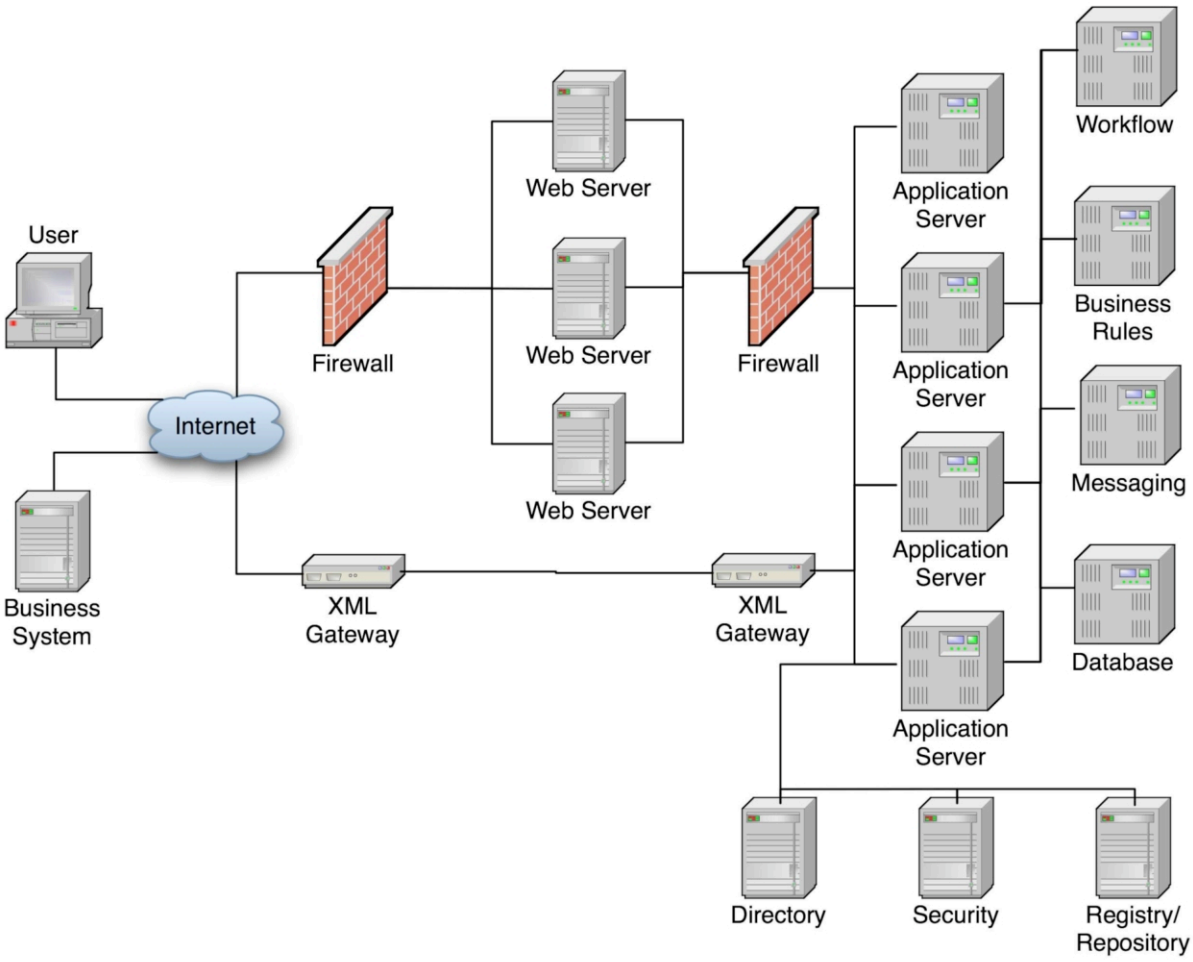
Rules Engine is the infrastructure in the J2EE platform that enables the components to be configured according to the business needs. Enterprise Java Bean components that perform most of the business functionality can be configured to base business operations according to rules specified through a rules engine. Most of the common rules engines have rules editor to input rules and XML based rules configuration and access APIs to programmatically read the rules. Some of the most popular rules engines are ILOG JRules and Advisor Blaze rules engine.

4.3.9 Email Server

The Email server is the infrastructure that provides a centralized location for the exchange of information through the sending and receiving of mail messages. In the J2EE platform JavaMail API is used to interact with the Messaging/Email servers. Some of the collaboration services like scheduling meeting, appointments and calendar services can also be accomplished using email/messaging servers. Some of the most popular email servers are the Sun ONE messaging server and the Lotus server.

4.4 Deployment View

The Deployment view demonstrates the network infrastructure view of the physical elements in the ITA. This infrastructure configuration aims to address some of the important considerations for systemic qualities (QoS), such as security, availability and scalability. The following section elaborates the major actors in the deployment view in perspective of the Systemic quality that they impact.



- **Security**

  Firewalls have been added to protect the trusted zone from the semi-trusted zone and the Internet.

  Security Server will provide the services for authenticating and authorizing users accessing the application or service.

- **Availability**

  Adding multiple instances of the Web Server, Application Server, and Database server provides fail over capabilities and targets high availability.

- **Scalability**

  Clustering the Web/Application Server enables load balancing and thus depicts horizontal scalability of the architecture.

- **Reliability**

  Standby and replication of the Database Server and Directory Server provides high reliability in the application architecture for the data integrity.

- **Maintainability**

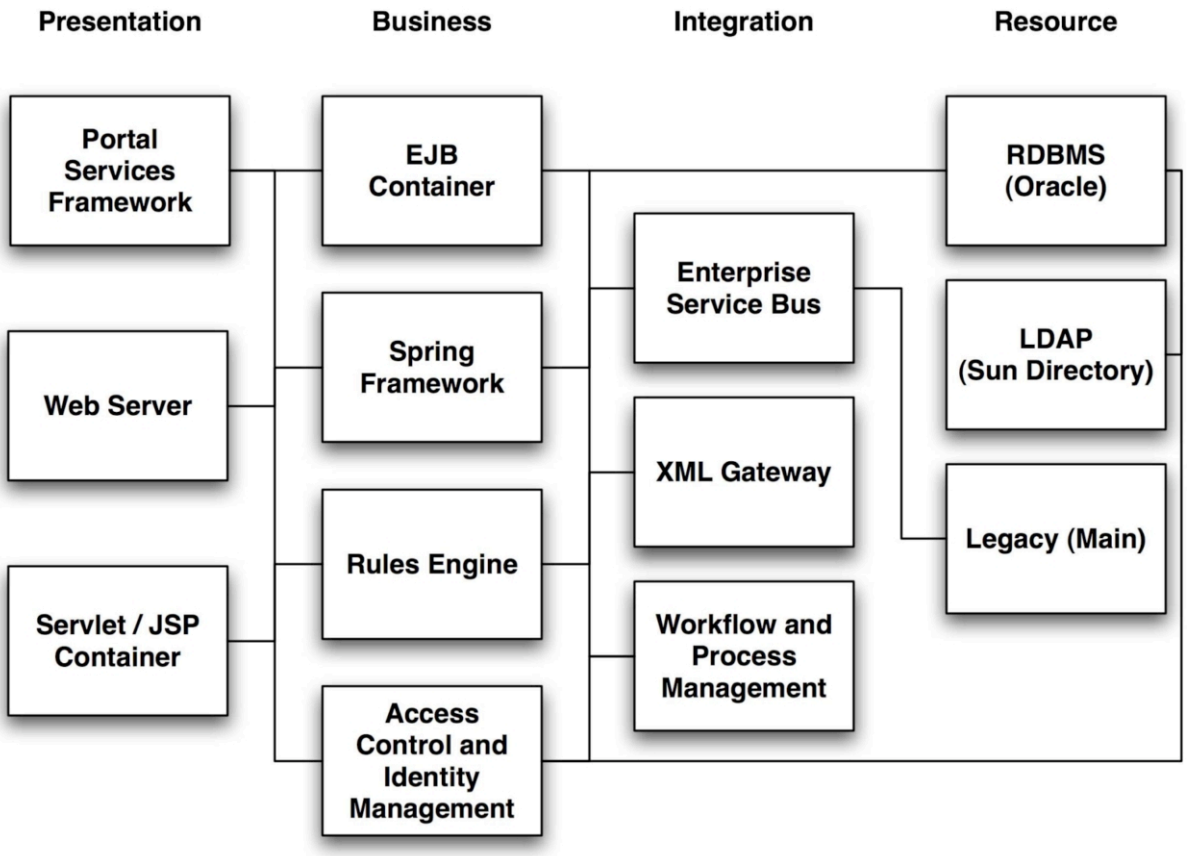
Most of the infrastructure platforms that are used in this architecture are open standards compliant (J2EE) contributing to the ease of maintenance of the application.

- **Manageability**

  The infrastructure platforms used in the architectural model (web/application servers) comes along with administration tools and utilities, which increases the manageability of applications deployed in them.

4.5 Platform View

The Platform view provides a snapshot of the various different infrastructure platforms in which the IT taxonomy groups will be deployed. It also separates out these infrastructure platforms into the Presentation, Business and Integration/Resources layers of the architecture. Most of the platforms identified in this view are candidates that have to undergo a process analyzing the industry acceptance of such products and prior experiences with them. The criteria will vary from product to product but mostly will focus on integration through open standards.



Each cube in the above figure represents a node, or physical box, with the main processes running on that node listed below the platform name in the cube.

Some key points about the platform view worth mentioning are:

All of the processes are significant enough that they have been allocated to their own node. Although this may not be implemented in an environment that is only used for development or functional testing, it is recommended for a production environment and should therefore also be replicated in environments intended for performance and stress testing. A System Test environment contains a scaled down set of processes described above. The System Test environment is used for functional testing and integration testing. A Quality Assurance (QA) environment is a replica of the production environment, perhaps at a smaller scale, and is where the stress and performance testing occurs on all services before they are promoted to production.

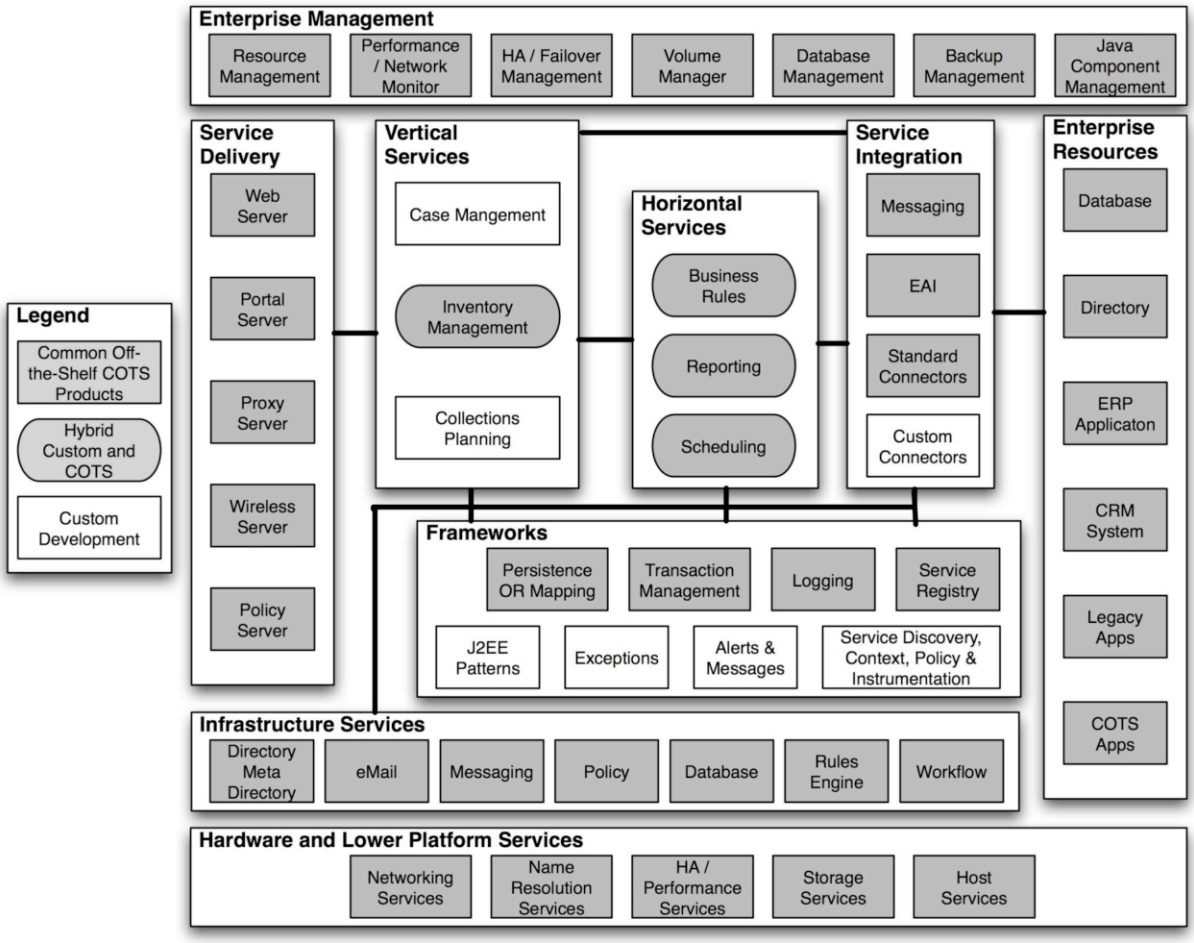
Although there may be multiple instances of a process running on given node and/or multiple instances of a given node within a particular environment, this information is not depicted in the platform view.

4.6 Component Realization View

The Component Realization View represents the various services and the frameworks represented in the conceptual architectural model and the respective realization mechanisms. The identified services and frameworks can be realized in three major ways.

- By using Commercial Off-The-Shelf (COTS) products
- By doing custom development over the COTS products to tailor them for HHS needs (HYBRID)
- By developing completely from scratch (CUSTOM)

The following figure classifies the different frameworks and services based on the above-identified three categories:



4.6.1 Commercial Off-the-Shelf (COTS) Products

Commercial Off-the-Shelf products include

- Hardware platform
- Operating system
- Storage devices
- Infrastructure platforms like the web server, application server, policy server, portal server, and database server
- Enterprise Information systems like the Mainframe systems
- Integration Systems like the DataPower XML Gateway.
- Third party tools like enterprise management tools (Tivoli, HP-OV)

These products provide out of the box functionality to address some major behavioral requirements of the conceptual architectural model. The figure illustrates that a majority of the identified services could be realized by procuring these COTS products, thus reducing the requirement to do custom development to the minimum possible.

4.6.2 Hybrid of Custom Development and COTS Product

There are some of the services that are specific to the business requirements of HHS, such as a service tracking system, that do not come as an out-of-the-box feature of a COTS product. Most COTS products provide programmable extensions to allow additions and tailoring required meeting specific business requirements of a customer. The Hybrid strategy is to make use of such extensions and tailor a commercially available COTS product to fit business requirements, thus minimizing the developmental effort in doing it from scratch.

4.6.3 Custom Development

Certain frameworks like the J2EE patterns and best practices, messaging and access control services are examples of standards that are to be enforced across HHS as an organizational practice.

Some of the HHS-specific functions like fine-grain access control can be used horizontally across applications. These functions may need to be custom-developed so that they provide necessary reusable interfaces. Connectors and APIs to external applications are required to establish communication from services that are to be deployed in a J2EE application server.

All of these frameworks and services will be custom-developed using industry-adopted standards/products/APIs to achieve reusability and compliance with the ITA.

4.7 Application Architecture views

This section describes the different application level views that EOHHS initiatives will be expected to provide in order to be compliant with the ITA as well as the EOHHS Project Methodology. Each of these architectural views is presented below along with examples.

4.7.1 Structure View

When custom development is involved, this view describes architecturally significant packages and static dependencies among them. Package responsibility and source (custom, reused, COTS, etc.) should be evident. The chosen decomposition should be elaborated and justified in terms of well-defined decomposition principles such as layering, distribution, generality, etc. Since this may involve several levels of detail, it may be desirable to begin with a diagram outlining the breakdown into layers and possibly tiers, and then follow with more detailed breakdowns into areas with relevant complexity. Here is an example package of a top-level package diagram showing dependencies:

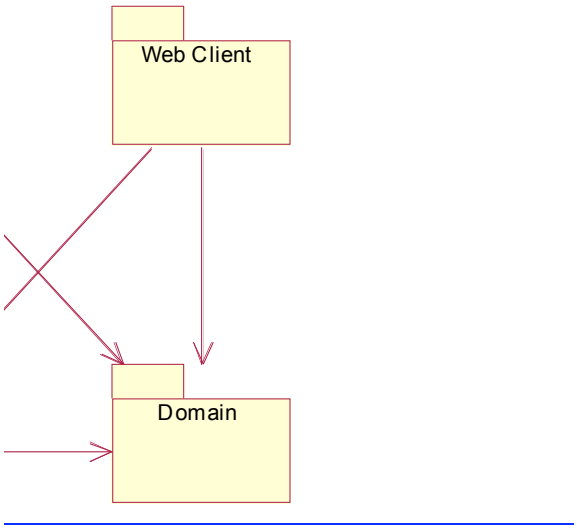


Figure 4.5: Sample Structure View

4.7.2 Configurations View

This view describes different application configurations in terms of components, their dynamic dependencies, and their physical positioning. Component diagrams, optionally overlaid on deployment diagrams, can be employed for this purpose. If sufficiently distinct, separate subheadings can be provided for the distinct configurations such as development, system test, QA and production. Note that lower-layer details will be captured later in their own views, although configurations here should be consistent with those later views. Below is an example component diagram focusing on dynamic dependencies:

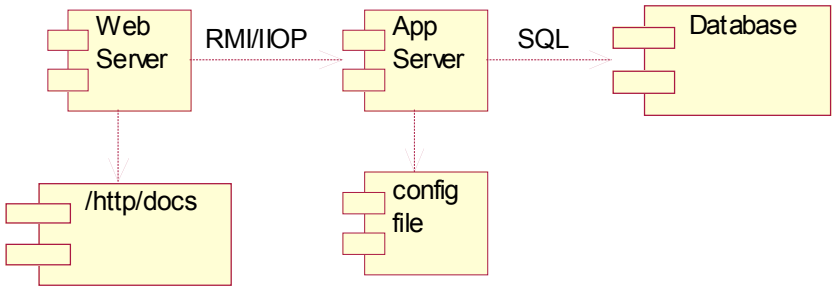


Figure 4.6: Sample Configuration View

4.7.3 Behavior View

This view describes dynamic interactions fulfilling use case functionality. The selected interactions should be representative but not exhaustive. Representative interactions describe architecturally significant detail that is best expressed in terms of message flows over time, in contrast with the previous

sections that portray static relationships. Sequence diagrams may be employed for this purpose. This is an example sequence diagram:

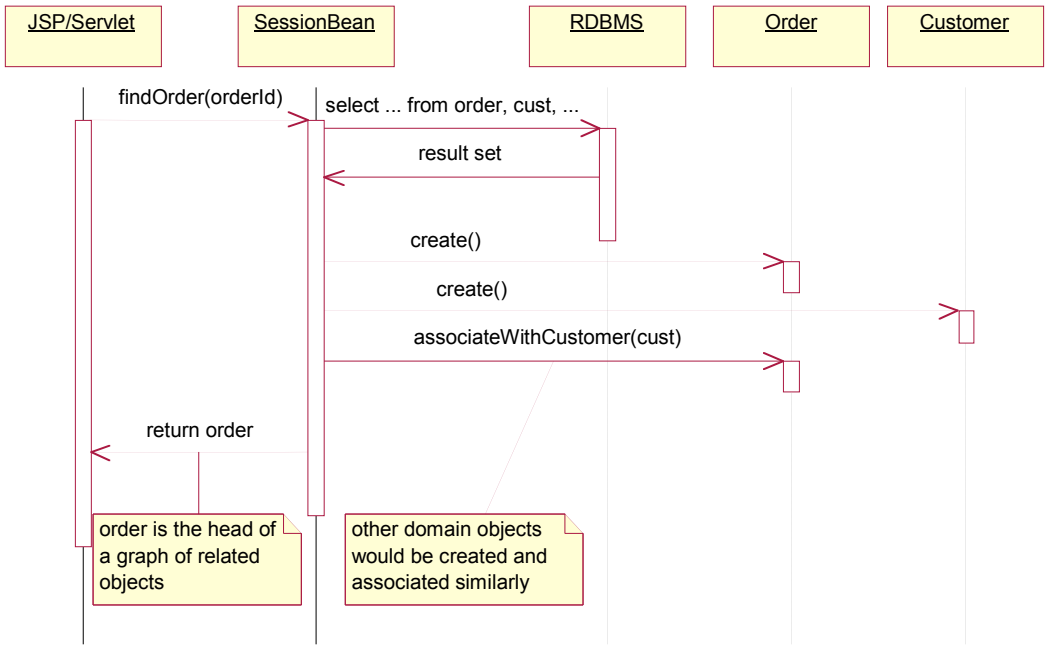


Figure 1.7: Sample Behavior View

4.7.4 Selected Technologies View

This view describes each major mechanism required by the system. This should include a list of mechanisms, their definitions and why they are being incorporated into the application. Additionally, a table may be used to outline the tier, container, and platform in which the mechanism runs, and the Application Programming Interface and Management Interface through which the mechanism is accessed and managed. Included here is a sample table.

Also, this view describes any of the mechanisms outlined in the previous section that must be custom built. Static and dynamic UML diagrams can be employed as necessary for this purpose. Both how they should be used and how they are architected should be covered.

Tier	Mechanism	Provider	Platform	API	MI
Presentation	Session management	Apache Tomcat Web Server	J2EE	Servlet container	Management Console
	HTTP protocol conversion			Servlet	
	Load balancing		Sun Plugin		HPOV
Business Logic	Connection pooling	JBoss Application Server	J2EE	JDBC	JMX, HPOV
	Transaction control			Enterprise JavaBeans	JMX, HPOV
	Auditing	Sun	IdM	IdM and AM SDK	HPOV

	Guaranteed messaging	MQ/Series	J2EE	JMS	
Resource	Persistence	Oracle 9I	J2EE	JDBC/SQL	Oracle Enterprise Manager
	Naming	Directory Server	JES	JNDI/LDAP	

Table 4.1: Sample Incorporated Mechanisms

4.7.5 Lower Configurations View

Describes configurations below the application layer, in direct or indirect support of the application layer. Includes detail not directly evident at the upper platform layer, such as DHCP servers or lower platform level fail-over capabilities. Describes the hardware configuration(s) including processing nodes, routers, firewalls, etc. Deployment diagrams can be used to describe nodes and network communication paths. What this might look like is:

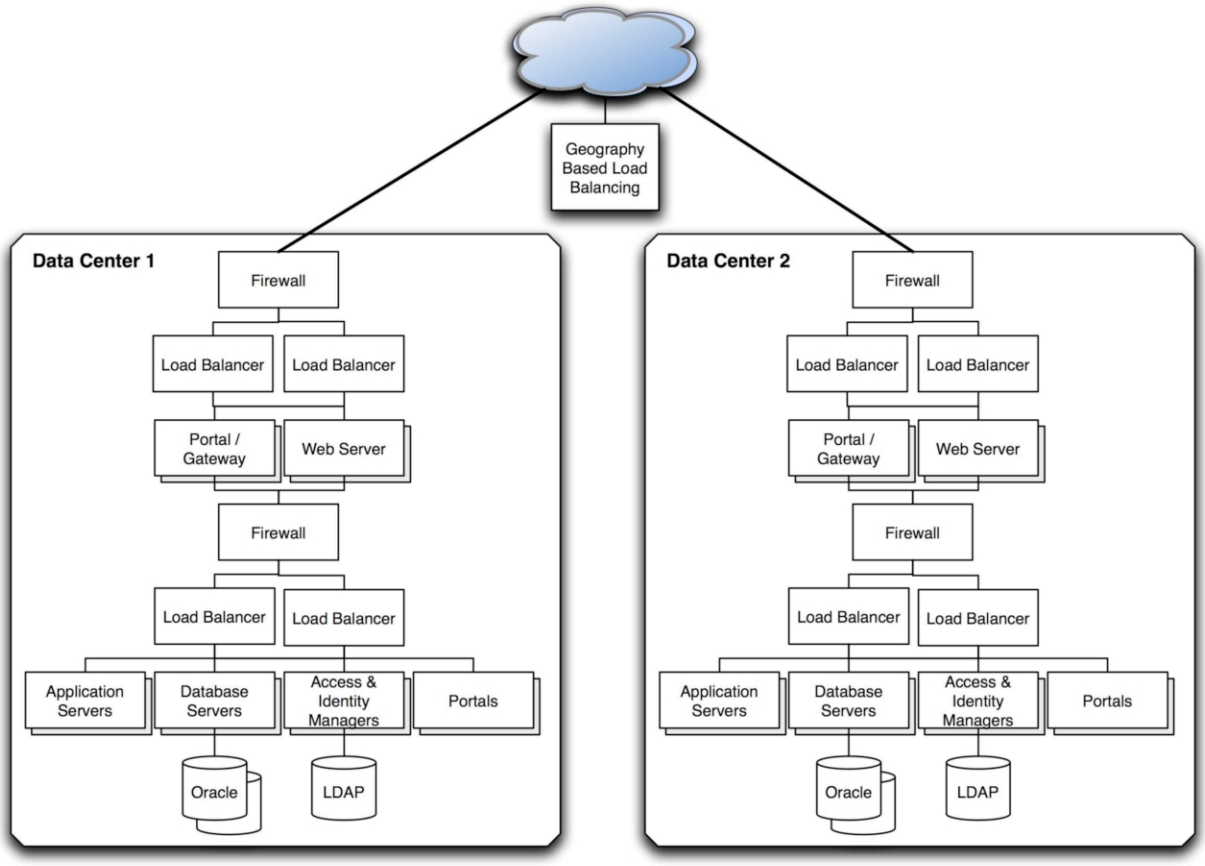


Figure 4.8: Sample Configuration View

4.7.6 Systemic Qualities View

In the nonfunctional requirements view all of the layers and tiers of a system are tied together in a unifying way and related to the primary requirements for the architecture, the nonfunctional

requirements. Matrices demonstrating the many dimensions can be employed to provide a summary view, and/or textual descriptions can provide more detail. Each of the following topics should be addressed for each nonfunctional requirement: direct and derived requirements, solution, risk validation strategy, and future evolution. An example matrix that describes the highest priority "ilities" for a specific project is provided:

Layer	Components	Security	Throughput Scalability	Reliability Availability	Manageability
Application	J2EE 1.4 based	UID & password	Minimize state, remote components	Replicated session, loose coupling	Common LDAP user records
Upper Platform	JBoss App Server	SSL, LDAP, ACLs	Resource pooling, clustering	Dual LDAP	Remote SNMP administration
Lower Platform	Linux Red Hat 9	Server lockdown	Load balancing	Clustering	Remote SNMP administration
Hardware	Intel	Physical isolation, firewalls	Expandable number of processors	Redundant networking interfaces	Remote SNMP administration

Table 4.2: Sample Nonfunctional Requirements View

– End of Document –